TP 5.04
Authentification et Anti-Piratage par les filtres

Objectifs

Savoir réaliser un site avec authentification et protections élémentaires.

Préambule

Dans cet exercice, on souhaite créer un site minimal dont certaines pages sont soumises à authentification.

Q1. Créez un projet Maven `tp504`,

— prototype: `maven-archetype-webapp`

— groupid: `fr.but3`

```
mvn archetype:generate -DarchetypeArtifactId=maven-archetype-webapp -DgroupId=fr.but3 -DartifactId=tp504
```

Q2. Ajoutez au pom les propriétés classiques (encodage, version de compil). Ajoutez aussi les dépendances: `jakarta-servlet`, `postgresql` ainsi que le plugin `cargo-maven3-plugin`

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>fr.but3</groupId>
  <artifactId>tp504</artifactId>
  <packaging>war</packaging>
  <version>1.0-SNAPSHOT</version>
  <name>tp504 Maven Webapp</name>
  <url>http://maven.apache.org</url>
  <properties>
    <maven.compiler.source>17</maven.compiler.source>
    <maven.compiler.target>17</maven.compiler.target>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <cargo.maven.containerId>tomcat10x</cargo.maven.containerId>
  </properties>
  <dependencies>
    <!-- ====================== -->
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
    </dependency>
    <!-- ====================== -->
    <dependency>
      <groupId>org.postgresql</groupId>
      <artifactId>postgresql</artifactId>
      <version>42.6.0</version>
    </dependency>
    <!-- ====================== -->
    <dependency>
      <groupId>jakarta.servlet</groupId>
      <artifactId>jakarta.servlet-api</artifactId>
      <version>6.0.0</version>
      <scope>provided</scope>
    </dependency>
    <!-- ====================== -->
  </dependencies>
  <!-- ====================== -->
  <build>
    <finalName>tp504</finalName>
    <plugins>
```

```
        <plugin>
          <groupId>org.codehaus.cargo</groupId>
          <artifactId>cargo-maven3-plugin</artifactId>
    <version>1.10.9</version>
        </plugin>
        <!-- A ajouter à cause de la version Maven < 3.8 -->
        <plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-war-plugin</artifactId>
    <version>3.4.0</version>
        </plugin>
      </plugins>
    </build>

  </project>
```

Q3. A ce stade vous devez déjà pouvoir lancer le serveur et accéder au `Hello World!` par défaut. Conservez le, c'est un test parfois utile !

Exercice 1 : Les JSP (45mn)

On souhaite avoir une page JSP qui affiche le message

`Vous avez accédé x fois à cette page sur les y au total`

Les hashmap ne stockent que des objets. Il est donc necessaire d'"encapsuler" l'entier `int` dans une classe. La classe wrapper `Integer` pourrait être candidate, mais malheureusement elle est immutable, donc impossible à incrémenter, ce qui la rend peu pratique pour un tel compteur.

Q1. Ecrire une JSP `cpt1.jsp` qui utilise pour cela la classe `AtomicInteger`

Q2. Ecrire une JSP `cpt2.jsp` qui utilise pour cela une classe interne `Cpt`.

```
public class Cpt
{   private int val=0;
    public int intValue() {return val; }
    public String toString() {return ""+val; }
    public void incr() {val++;}
}
```

Q3. Ecrire une JSP `cpt3.jsp` qui utilise pour cela la même classe `Cpt` mais externe cette fois.

Q4. Ecrire une JSP `cpt4.jsp` qui accès à l'objet via une balise `useBean` et des EL expressions.
Selon les versions de Tomcat les EL sont autorisées par défaut ou pas (tomcat 10).

```
<%@ page isELIgnored="false" %>
```

Exercice 2 : Les bases d'un système d'authentification (45mn)

Q1. Dans `webapp` créez un répertoire `private` , puis dedans une JSP `page1.jsp` qui affiche un simple message de bienvenue (A terme, il faudra s'authentifier pour y accéder).

Q2. Créez maintenant une page `login.jsp` contenant un formulaire login/mdp qui appelle une servlet `Verif.java`. Cette servlet teste les valeurs du login/mdp par un simple `equals`. Si elles sont conformes aux attentes (votre nom/prénom par exemple) , elle stocke dans la session un token avec le login récupéré puis redirige vers `private/page1.jsp`. Si le login/mdp n'est pas conforme, `Verif` redirige vers `login.jsp` avec un message d'erreur en paramètre.

Q3. Modifiez `login.jsp` pour qu'il affiche le message d'erreur si il existe, et modifiez `page1.jsp` pour qu'elle affiche le nom de la personne logguée dans son message obtenu par session.

Q4. Testez le bon fonctionnement de ce mécanisme avec un login incorrect (on retourne au login et un message d'erreur s'affiche), ainsi qu'avec un login correct (on arrive sur `page1.jsp` et le login s'affiche.)

Q5. Ajoutez maintenant un filtre `MyAuthentFilter` sur l'url `/private` qui transmet la requête vers la servlet appelée si le token est présent dans la session, et redirige vers la page de login dans le cas contraire. Juste avant cette redirection,

on stocke dans la session une clé `origine` avec l'URI appelée, de manière à ce que `login.jsp` puisse maintenant rediriger vers cette origine quand elle est fournie.

A ce stade, au premier appel de `page1.jsp`, un login/mdp doit être saisi. Si on recharge la page, on ne repasse plus par le login.

Q6. La force des filtres : Ajoutez une JSP `private/page2.jsp`. Détruire le cookie, et appelez directement cette page. La page de login doit automatiquement être appelée, et la redirection doit se faire correctement sur `page2`.
Quelle que soit la page placée dans `private`, son appel necessitera un login/mdp et on sera directement ramené à cette page.

Si au bout de 45mn vous n'avez toujours pas réussi, vous trouverez une correction sur Moodle (`tp504.zip`).

Exercice 3 : Un système d'authentification plus évolué (1h00)

Q1. On souhaite améliorer le processus d'identification en allant chercher les login/mdp dans une base de données. Créez une table `utilisateurs(login,mdp,dat,ip)`.

Q2. Modifiez `Verif` pour qu'elle teste cette fois dans la base de données si l'utilisateur est présent ou non.
On utilisera pour cela une requête sur `preparedStatement` avec fabrication de l'URL par concaténation des login/mdp : `"select * from utilisateurs where login='"`login+"' and mdp='"+mdp+"';"+

Q3. Améliorer `Verif.java` et `page1;jsp` en faisant en sorte que, à chaque visite d'un utilisateur, on mette à jour sa date de connexion (objet `java.util.Date`) et l'adresse IP de la machine qu'il utilise (méthode `getRemoteAddr`).
Les anciennes valeurs seront utilisées pour afficher le message

```
Bonjour Mr xxx, bienvenu sur ce site
Bonjour Mr xxx, votre dernière connexion date de hh:mm:ss de la machine 0.0.0.0
```

Q4. Modifiez votre code pour que les mdp soient maintenant hachés en `md5`. Postgres possède nativement une fonction `md5(text)` que vous pouvez utiliser dans vos requêtes.
Par exemple pour mettre à jour la base: `update utilisateurs set mdp=shamd5(mdp);`

Exercice 4 : Eviter les injections (1/2h)

Pour l'instant nous n'avons volontairement pas lutté contre les différentes injections. Dans `Verif` la requête est écrite en concaténant des chaines de caractère.

Q1. Essayez de vous connecter au site par injection SQL.
On pourra notamment tester le classique ' OR '1'='1

Q2. Corrigez `Verif` pour éviter l'injection SQL.

Q3. Modifiez `Page2.jsp` pour que l'on puisse lui passer un paramètre qu'elle affichera.

Q4. Testez une injection XSS.
On pourra notamment tester le classique `<script>location='http://www.libe.fr'</script>`

Q5. Développez un filtre `MyXSSFilter.java` qui filtre toutes les requêtes possibles. En utilisant un `RequestWrapper`, redéfinissez la méthode `getParameter` pour qu'elle remplace systématique les < par des `<` ,les > par des `>`, et qu'elle supprime les mots `script` .